An Empirical Evaluation of Virtual Circuit Holding Time Policies in IP-over-ATM Networks

S. Keshav, Carsten Lund, Steven Phillips, Nick Reingold, Huzur Saran

Abstract— **When carrying Internet Protocol (IP) traffic over an Asynchronous Transfer Mode (ATM) network, the ATM adaptation layer must determine how long to hold a virtual circuit opened to carry an IP datagram. In this paper we present a formal statement of the problem and carry out a detailed empirical examination of various holding time policies taking into account the issue of network pricing. We offer solutions for two natural pricing models, the first being a likely pricing model of future ATM networks, while the second is based on characteristics of current networks. For each pricing model, we study a variety of simple non-adaptive policies as well as easy to implement policies that adapt to the characteristics of the IP traffic. We simulate our policies on actual network traffic, and find that policies based on LRU perform well, although the best adaptive policies provide a significant improvement over LRU.**

Keywords— **ATM, IP, virtual circuits, holding policies, adaptive policies, paging, inter-arrival time distribution.**

I. Introduction

It is generally accepted that, in the near future, large computer networks will be connection-oriented, with at least the data-link layer connectivity being provided by Asynchronous Transfer Mode (ATM). These networks will need to communicate with existing networks. The world's largest computer network, the Internet, with more than a million computers, uses the connectionless Internet Protocol (IP). For the huge existing investment in IP networks to remain useful, we must devise mechanisms to carry IP traffic over ATM networks. A fundamental issue is how to carry datagrams over virtual circuits. It is clear that the arrival of an IP datagram should cause a virtual circuit to be opened, if one is not open already. However, it isn't clear how to handle the open circuit thereafter. It would be desirable to keep it open for some time, to amortize the cost of opening the circuit over many packets. On the other hand, if no more packets will arrive soon, it is better to close the connection. The ATM adaptation layer must decide heuristically how long to hold the circuit open, since the IP datagrams do not contain information about the length and rate of any higher layer conversations. Similar problems arise in carrying IP traffic over other connection-based networks, such as X.25. In this paper, we present an empirical study of the arrival process of IP datagrams to the ATM adaptation layer. We find that the data shows temporal locality of reference, and therefore holding time policies based on Least Recently Used (LRU) perform well. However, we obtain a significant improvement over LRU by using adaptive policies that conform to the inter-arrival time distribution of each conversation.

Keshav, Lund, Phillips, and Reingold are with AT&T Bell Laboratories, Murray Hill, NJ 07974, USA. Keshav and Saran are with Department of Computer Science and Engineering, Indian Institute of Technology, Hauz Khas, New Delhi 110016, India

In the next section we discuss previous work in this area. Section III presents the necessary background and details of the network pricing models. In Section IV we describe and analyze the empirical data used in this study, while classes of holding time policies are discussed in Section V. Section VI presents the policies for a pricing model with holding costs, together with a comprehensive comparison of the policies. Section VII presents the holding policies for a pricing model with a maximum number of connections, together with an empirical evaluation of their performance. Finally, Section IX closes with some discussion and conclusions.

II. Previous Work

The holding time problem arises naturally in carrying connectionless protocols such as IP over connection-oriented networks such as X.25 and Datakit. While existing implementations embody several holding time policies such as Least Recently Used [1], a formal statement of the holding time problem and a comparative study of these policies was presented by Saran and Keshav [2] and further studied by Lund, Phillips and Reingold [3].

Lund, Phillips and Reingold [4] gave a theoretical treatment of the pricing model with a maximum number of connections, described below. Their theoretical algorithm is the basis of one of the adaptive algorithms studied in this paper. Harita and Leslie [5] studied the related problem of dynamically allocating bandwidth when carrying ATM on a narrowband ISDN network.

III. Background and Pricing Models

The most important factor in determining a virtual circuit holding time policy is the pricing model of the network. The pricing model determines which parameters the VC holding time policy should seek to minimize. We study two pricing models: the first is a possible pricing model of future ATM networks, while the second is based on characteristics of current networks.

In both cases, we assume that IP packets are partitioned into *conversations* based on their source and destination Internet addresses. (The ATM adaptation layer may offer a finer grained connectivity by further partitioning conversations based on the port number of the IP source and destination.) Throughout the paper we will simplify the presentation by the following slight abuse of terminology: we often refer to a conversation as a *circuit* that can be repeatedly opened for transmission of a packet, and closed sometime thereafter. In actuality a new virtual circuit must be set up each time.

There are additional issues that might arise when carrying IP traffic over an ATM network, such as deciding what bandwidth to request when opening a VC to transmit an IP datagram [5]. If insufficient bandwidth is requested or available from the ATM network, queueing of IP datagrams at the IP-ATM interface would become necessary. We do not consider such issues in this paper, instead only focusing on the problem of finding good holding time policies.

In the rest of this section we describe the two pricing models.

A. A Pricing Model with Holding Costs

Future ATM networks are expected to support a large number of virtual circuits that will be available to end-users on a pay-per-use basis. The manner in which users are charged is likely to be analogous to telephone billing, hence we study the following pricing model: there is a call connect charge of C monetary units, and a holding cost of H monetary units per time unit that a circuit remains open. The holding cost serves as an incentive for a user to return unused resources to the network. For convenience we assume that monetary units are scaled so that $H = 1$. There may be other charges associated with a circuit, such as a per-packet usage charge, but these charges do not affect the choice of virtual circuit holding policy.

Setting up a virtual circuit involves both a financial cost of the call setup and a user delay waiting for the call connect. To quantify the loss of utility to the user due to the call connect delay we define an *open cost*, $\mathcal{O}$, measured in monetary units, that is an estimate of the combined financial and user cost of a call setup. A system manager can vary the value of $\mathcal{O}$ to reach a satisfactory price/performance tradeoff.

After each packet arrives on a circuit, we must decide how long to keep the circuit open. This length of time is the *timeout* — if no packet arrives before the timeout, the circuit is closed and must be reopened when a packet eventually arrives. Keeping a circuit open too long results in a large holding cost, while closing it too early results in an unnecessary open cost. Thus the problem that the ATM adaptation layer must solve is to determine a timeout that incurs a low cost.

B. A Pricing Model with a Maximum Number of Connections

Some traditional virtual circuit oriented networks regard virtual circuits as a valuable resource, and have a limit on the number of virtual circuits that an end-user may have open simultaneously. This is often true of X.25 networks [1], where the limit is typically between 32 and 128. For these networks we use the following pricing model: the user pays a fixed charge for a block of connections to a site, and is then charged for each call setup. Since there is no cost for holding a circuit open, it makes sense for the user to keep the maximum number of connections open all the time. As observed in [1] this pricing model is closely related to *paging*: the connections are analogous to page slots in memory, while the conversations that are competing for the connections are analogous to the pages of virtual memory. When a packet arrives on a virtual circuit that is closed, the virtual circuit must be opened to transmit the packet, and some other virtual circuit must be closed to satisfy the bound on the number of connections. This corresponds to a page fault in the paging problem.

The cost of a call setup might include a loss of utility to the user due to the time delay in performing the call setup. In any event, the quantity we wish to minimize is the number of call setups.

IV. Workload Analysis

We collected traces of packet arrivals from Ethernet networks, using the SunOS `etherfind` command. This command places the Ethernet interface in promiscuous mode and collects all the Ethernet headers received on the board, along with time-stamps. The command was run on Ethernets at AT&T Bell Laboratories in Murray Hill, the University of California at Berkeley, the University of Southern California in Los Angeles, Yale University in New Haven, and the Indian Institute of Technology in Delhi. We collected a total of 53 traces, each consisting of between 2000 and 20000 packets, with broadcast packets filtered out. The five networks all cater to research communities, but offer widely varying computing environments, ranging from primarily PCs in Delhi to high-performance workstations at Berkeley.

The five environments had quite distinct characteristics: the data from UCB and USC were taken from LANs with a large number of active workstations, and there were many simultaneously active conversations. The data from IIT Delhi were taken from a LAN that had a few workstations and a number of PCs using TCP/IP. The number of active connections here was significantly lower and the data consisted of a smaller number of connections being sampled for a larger period of time. The AT&T Bell Labs and Yale data were taken from networks with a small number of active workstations and had somewhat similar characteristics to the IIT Delhi data.

In gathering traces from LANs, we are assuming that this traffic will actually be carried over a WAN. This assumption may seem surprising at first, since current LAN and WAN traffic characteristics differ widely. However, we anticipate that as high speed ATM WANs become available, higher throughputs and lower delays will significantly alter wide area traffic patterns. Given high speed wide area networks, it is feasible to mount remote file systems (NFS), and run client server applications (such as X) over a WAN, whereas these options are not common today.

A. Data Analysis

Based on the application level characterization work by Caceres et al [6], our intuition was that conversations in the traffic traces we collected would show behavior on widely varying time-scales, including a *user time-scale* and a *network time-scale*. The idea is that some usage of the network must be mediated by a human user, and thus shows somewhat larger inter-arrival times, while other usage is mediated directly by a computer, and so will have shorter inter-arrival times. As an example, during an FTP session, a human user may type "`get` *filename*", where each keystroke is at the user time-scale. However, the response, which (in an uncongested network) is a stream of back-to-back packets, would be at the network time-scale, since a user would usually not generate packets at that speed. Similarly a Mosaic session may involve a burst of activity when a user follows a hyper-link, followed by a period of inactivity while the user digests the received information.

Results in Paxson and Floyd's recent paper [7] empirically confirm our intuition about the existence of user and network time scales. Specifically, they found that both Telnet and FTP data packet arrivals are bursty over multiple time scales. At the slower time scales, these correspond to user interactions (typing, or FTP `get` commands), and at faster time scales, these reflect network dynamics.

These observations are reflected in the packet traces we collected. Figure 1 shows a representative trace from the data we use. Each line is a simplex conversation between a pair of Internet hosts, and each diamond represents a packet. Different conversations have very different inter-arrival characteristics, and there is also variation in inter-arrival times inside a conversation. Incidentally, in Figure 1, the reader may notice that some pairs of conversations are correlated; such pairs correspond to duplex conversations.

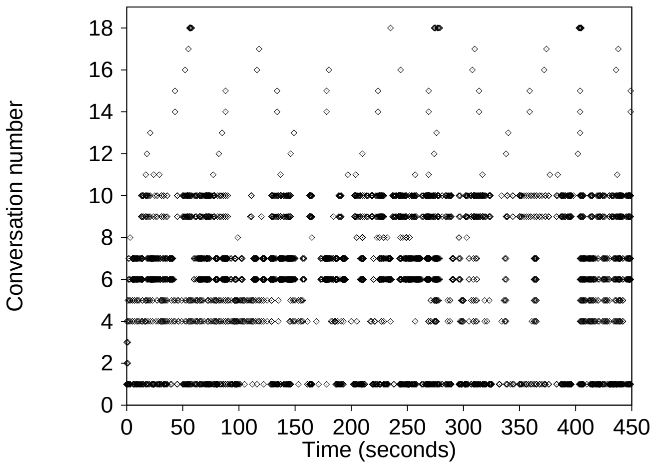

Fig. 1. Conversations with different inter-arrival distributions.

The observed inter-arrival distribution of conversation number 10 is shown in Figure 2. This is a bursty distribution, that displays the clustering phenomenon described above. Under most reasonable pricing policies, the best way to handle such a conversation is to hold its connection open while the inter-arrival times are drawn from the faster time-scale and drop the connection at the end of a burst, when we anticipate that the next inter-arrival time will be drawn from the second-level time scale.

On the other hand, conversation number 4 displays very different characteristics. This conversation has a lot of traffic, but does not display burstiness at this time scale, see Figure 3. (However, at a larger time scale this conversation may also appear bursty.) Another non-bursty distribution is represented by conversation 12, which has fairly regular inter-arrival times on a much larger time scale than conversations 4 or 10.

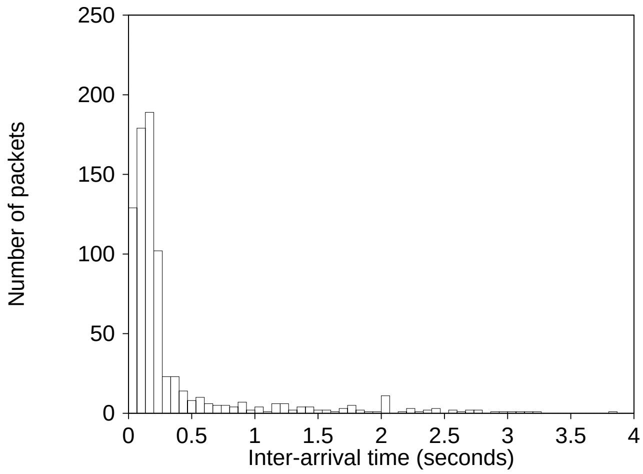

Fig. 2. Inter-arrival time distribution for a bursty conversation (VC 10 from Figure 1).

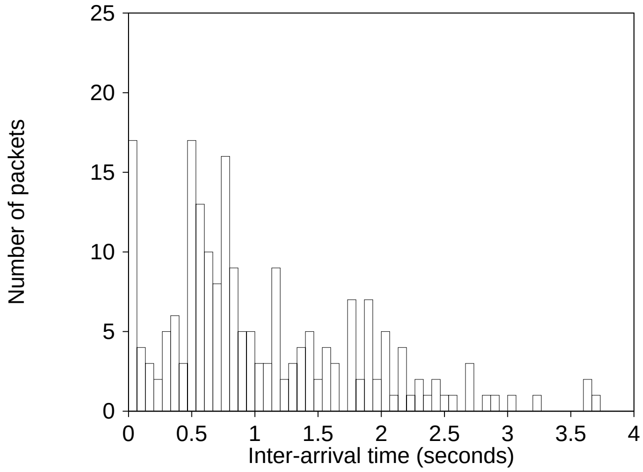

Fig. 3. Inter-arrival time distribution for a less bursty conversation (number 4 from Figure 1).

There are two observations to be made from examining the data. Firstly, across all conversations in a trace there is *locality of reference*, i.e. the next packet to arrive is likely to be in a conversation that has recently had a packet. Secondly, individual conversations have characteristics that remain fairly consistent for periods of time. These two observations are discussed in more depth in the following sections.

B. Temporal Locality

To look for temporal locality, we looked at the frequency of reference to a least recently used stack corresponding to a trace. We built a small simulator that looked at a trace, and pulled each reference to a conversation to the top of a stack. We also kept track of the number of references to each level of the stack. If our hypothesis about temporal locality is true, then the frequency of references to the top of the stack would be much higher than the frequency of references to the lower levels of the stack. Indeed, all our data show a steep decline in the frequency of reference to a stack level as the depth increases (see Figure 4 for two sample traces), clearly indicating the presence of temporal locality. So, if at some time a conversation has been recently referenced, it is likely that it will soon be referenced again.

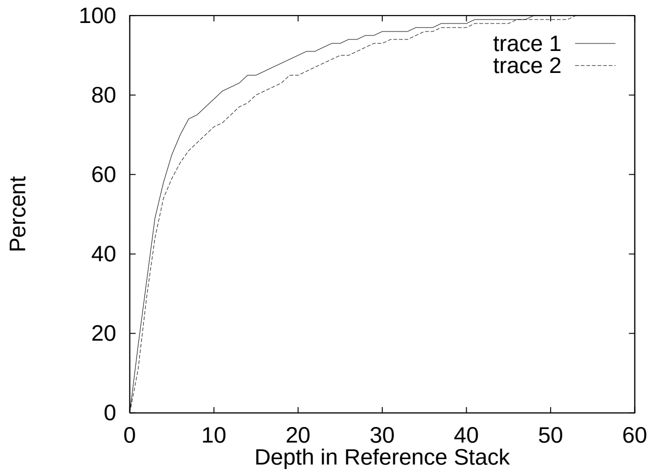

Fig. 4. Cumulative references vs. LRU stack depth

C. Consistent Behavior of a Conversation for Extended Time Periods

Different conversations can have widely different characteristics, in terms of bandwidth, regularity, burstiness, or other measures. For example, a telnet session involving a user entering text in an editor will have fairly regular inter-arrival times, corresponding to user keystrokes, with occasional bursts when the editor reformats the display. In contrast, an FTP session will be much more bursty, with less activity at the user time scale. If we plot a histogram of the inter-arrival times between packets in such a session, we expect to see a bursty distribution as depicted in Figure 2.

However a single session typically has consistent behavior over extended periods. For example, in our telnet example, the conversation will remain consistent as long as the user is entering text. Although the inter-arrival time distribution does not capture aspects of a conversation such as correlations between adjacent inter-arrival times, it provides a good method for predicting future inter-arrival times.

How can we use this observed behavior? One possible approach to developing adaptive holding policies is to construct a policy that works well against data generated according to some model of network traffic. For a model to be useful, it must allow for the wide variation in traffic observed in practice. However traditional models tend to be over-parameterized, while the self-similar stochastic model of [8] has parameters that seem computationally difficult to estimate.

Instead of assuming a model for the data, our adaptive policies make the single assumption that the inter-arrival time till the next packet in a conversation is likely to be drawn from the same distribution as the inter-arrival times that have been observed so far in that conversation. Thus we can use the observed inter-arrival distribution to make a good choice of timeout (in the holding cost model) or of which circuit to close (in the paging model). We make no assumptions about the structure of the inter-arrival time distributions, so our work is not based on any strict assumptions about the kind of traffic that will appear on future networks.

V. Holding Time Policies

The simplest holding policy is not to hold a VC at all. That is, on every packet arrival, a VC is opened, and then closed. Given the round-trip-time delay in opening a circuit and the cost of call setup, this option is not particularly effective.

The optimal holding policy is one that is non-causal, that is, it knows about the future. The specific optimal strategy depends on the pricing policy and system constraints. For example, in the paging model, the optimal strategy is to simply drop the VC on which a packet will arrive the farthest in the future. While the optimal policy is unachievable, it provides a benchmark against which to compare all other policies.

In general, a holding policy gathers some statistics about the inter-arrival times and uses these statistics to decide when each VC should be closed. For each pricing model, we consider several different holding policies. The policies differ in how much, and what kind of information is gathered about the inter-arrival time distributions for each VC.

In each pricing model the simplest policies we consider use no information at all about the observed inter-arrival times. We also consider a policy that maintains an exponentially averaged mean and deviation for each VC. One would expect that such a policy should be able to use this information to make better decisions regarding the closing of virtual circuits. However, our results indicate that this information is insufficient to design good holding time policies.

The best policies in each pricing model are based on gathering more complicated statistics for each VC. Essentially, an approximation to the entire inter-arrival time distribution is kept and updated each time a packet arrives. We call these policies ADAPTIVE since the choice of holding time for a given VC adapts to the inter-arrival time distribution for that VC. We show how to maintain the approximate inter-arrival time distributions with very little overhead.

Our adaptive policies were first developed and tuned to optimize performance on data sets from Berkeley, USC, IITD, and AT&T. Afterwards, to ensure that the policies would perform well not only on the data for which they were tuned, the policies were run on a second set of data from AT&T and on the data from Yale. The simulation results for the new data sets are consistent with the results of the original data sets (see Subsections VI-C and VII-C). It is therefore reasonable to expect that our findings will remain true for general Internet traffic.

VI. The Holding Cost Pricing Model

In the first pricing model we study there is a holding cost of 1 monetary unit per time unit that a circuit remains open. Each time a circuit is opened there is an *open cost*, $\mathcal{O}$, that includes both the financial cost of a call setup and an estimate of the cost to the user of waiting for the call connect. A system manager can vary the value of $\mathcal{O}$ to reach a satisfactory price/performance tradeoff.

After each packet arrives on a circuit, we must decide how long to keep the circuit open. This length of time is the *timeout* — if no packet arrives before the timeout, the circuit is closed and must be reopened when a packet eventually arrives. Keeping a circuit open too long results in a large holding cost, while closing it too early results in an unnecessary open cost.

Note that the charging of each circuit is independent of other circuits. It is possible that packet arrivals on different circuits will sometimes be correlated, but we make no use of such possible correlations, and consider each circuit in isolation.

A. Holding policies

In this section we consider several different holding policies in detail. In the simplest policy, the same timeout is used for each VC, while in the others some information about the previously observed inter-arrival times is used to set a timeout for each VC.

The (non-causal) optimal policy for this pricing model is trivial — if the next arrival is more than $\mathcal{O}$ time units in the future, the VC is closed, else it is kept open. This policy guarantees the smallest possible cost for each conversation. Although this policy cannot be used in practice since it relies information about the future, it is convenient to use as a basis for comparison with other policies (see Section VI-C).

A.1 Lru-based policies

The temporal locality in our data motivated us to explore Lru-based policies for holding VCs. In the simplest version, if a conversation has been idle for time t then we predict that the next arrival will be ct time units in the future, where c is a constant. Notice that the relative predicted arrival times correspond to relative depths in the LRU stack. Applying this method of predicting future arrivals we get a very simple policy: drop a conversation if it has been idle for $\mathcal{O}/c$ time units. Thus this policy sets the same timeout, $\mathcal{O}/c$, for all conversations. This policy has another very nice property:

Lemma 1: The cost incurred when using the Lru-based policy with parameter c is no more than $\max(c, 1/c) + 1$ times the optimal cost.

To prove this, observe that the worst case input is one where the packets arrive $\mathcal{O}/c$ time units apart. The optimal cost of serving this sequence is $\min(\mathcal{O}, \mathcal{O}/c)$ per packet whereas the Lru-based policy spends $(\mathcal{O}/c + \mathcal{O})$. Thus, we can guarantee that the cost incurred is no more than $\max(c, 1/c) + 1$ times optimum. In our study we examined a range of different values for c.

A.2 The Mean-Variance policy

The next strategy we consider is to predict the arrival time of a particular VC based on a small amount of history. The estimation algorithm was derived from Jacobson's work on good estimators for round trip times [9]. We measure the inter-arrival time for each VC and compute an exponentially averaged mean and an exponentially averaged mean deviation from the mean. For a given VC, let t_k be the kth inter-arrival time. For some fixed parameter $0 < \alpha < 1$ (we used $\alpha = 0.1$), the estimate of the mean inter-arrival time, μ_{k+1}, and the deviation, σ_{k+1}, are computed as follows:

$$\begin{aligned}\mu_{k+1} &= \alpha t_k + (1-\alpha)\mu_k \\ \sigma_{k+1} &= \alpha|\mu_k - t_k| + (1-\alpha)\sigma_k\end{aligned}$$

When a packet arrives, we use the current estimates $\mu = \mu_{k+1}$ and $\sigma = \sigma_{k+1}$ to choose a timeout. It is very likely that a packet will arrive to the VC in the interval $[\mu - 2\sigma, \mu + 2\sigma]$. If $\mu - 2\sigma > \mathcal{O}$, then the VC should be closed immediately. Similarly, If $\mu + 2\sigma < \mathcal{O}$, then the VC should certainly be kept open, at least till $\mu + 2\sigma$. If $\mathcal{O}$ lies in the interval $[\mu - 2\sigma, \mu + 2\sigma]$, then we have to make some assumptions about how the probability mass is distributed within the interval $[\mu - 2\sigma, \mu + 2\sigma]$. We assume that the probability mass is concentrated around the mean and close the circuit if $\mu > \mathcal{O}$ and keep it open otherwise. Thus, if $\mu > \mathcal{O}$ the timeout is set to 0, otherwise it is set to $\min(\mathcal{O}, \max(\mu + 2\sigma, C))$, where C is is a cutoff parameter that ensures that for very high rate conversations we do keep the circuit open for a reasonable time. In our work, we chose $C = \mathcal{O}/5$. However, we have seen that the results are insensitive to choice of C in the range $\mathcal{O}/3$ to $\mathcal{O}/6$.

A.3 The Adaptive Policy

Our adaptive policy uses the single assumption that the inter-arrival time till the next packet in a conversation is drawn from the same distribution as the inter-arrival times that have been observed so far. Thus we can use the observed inter-arrival times to make a choice of timeout that is suited to the individual conversation. To design the best adaptive holding policy we need to decide first what information to gather about the inter-arrival time distribution, and second how to use that information to determine the best timeout, assuming the next inter-arrival time is drawn from the same distribution. The following sections describe the solutions to these problems.

When $\mathcal{D}$ is known

Assume that we know the distribution $\mathcal{D}$ on the next inter-arrival time. Suppose that $\mathcal{D}$ has a probability density function $f_{\mathcal{D}}$, so that if T is an inter-arrival time drawn from $\mathcal{D}$, then

$$\Pr[T \leq t] = \int_0^t f_{\mathcal{D}}(x)dx.$$

We wish to set a timeout $t_{\mathcal{D}}$ that minimizes the expected cost of the next packet. This cost is the time the circuit is held open before the next packet arrives, plus $\mathcal{O}$ if the circuit must be reopened for the next packet. If the timeout is set to t, then the expected cost of the next packet can be expressed as

$$C(t,\mathcal{D}) = \int_0^t x f_{\mathcal{D}}(x)dx \quad + \quad (t+\mathcal{O})\int_t^{\infty} f_{\mathcal{D}}(x)dx.$$

The first term is due to inter-arrival times of at most t, where the cost is just the holding time (i.e. the inter-arrival time). The second term results from inter-arrival times greater than t, where the cost is the holding time plus the open-cost, $(t+\mathcal{O})$.

We seek to minimize the expected cost of the next packet, so we merely choose the timeout $t_{\mathcal{D}}$ to be the value of t that minimizes $C(t,\mathcal{D})$. Notice that $t_{\mathcal{D}}$ is a parameter of the distribution function only, so the timeout is the same for each packet in the conversation. However, different conversations have different inter-arrival time distributions, so will have different timeouts. The timeout $t_{\mathcal{D}}$ is thus tuned to the characteristics of the particular conversation. This derivation for the optimal timeout has also been obtained in the context of spinning on a lock in a shared-memory multiprocessor [10].

When $\mathcal{D}$ must be learned

The previous section assumes that the distribution $\mathcal{D}$ of inter-arrival times is known. In reality, $\mathcal{D}$ is not known, and must be inferred by observation. Hence we keep a histogram $\mathcal{H}$ of observed inter-arrival times. For each $i \in 0\ldots m-1$, $\mathcal{H}(i)$ is the number of observed inter-arrival times in the interval $[\frac{i\times M}{m}, \frac{(i+1)\times M}{m})$, where the parameter m is the number of entries in $\mathcal{H}$ and M is the maximum timeout we will use. Experiments suggest that a good value for m is between 10 and 100 (see Section VI-C). The value of M is not critical, as long as it is at least $\mathcal{O}$; in our simulations we use $M=\mathcal{O}$.

We would like to set a timeout $t_{\mathcal{H}}$ that is a close approximation to $t_{\mathcal{D}}$. Let $n(\mathcal{H}) = \sum_{0\leq i<m}\mathcal{H}(i)$ be the number of inter-arrival times observed so far. Using $\mathcal{H}$ as an approximation to $\mathcal{D}$, we define the estimated cost of a timeout t analogously to $C(t,\mathcal{D})$:

$$EC(t,\mathcal{H}) \quad = \quad \sum_{0\leq i:\frac{(i+1)\times M}{m}\leq t} \frac{\mathcal{H}(i)}{n(\mathcal{H})}\frac{i\times M}{m}$$
$$+ \quad (t+\mathcal{O})\left(\sum_{i\leq m-1:\frac{(i+1)\times M}{m}>t}\frac{\mathcal{H}(i)}{n(\mathcal{H})}\right).$$

Notice that the true "cost" of samples in histogram i is between $\frac{\mathcal{H}(i)}{n(\mathcal{H})}$ and $\frac{\mathcal{H}(i+1)}{n(\mathcal{H})}$, depending on the exact inter-arrival times that were placed together in the bin. We have chosen $\frac{\mathcal{H}(i)}{n(\mathcal{H})}$ in the above formula, simply because this choice gave slightly better empirical results.

Let t' be the value of t that minimizes $EC(t,\mathcal{H})$. A natural strategy is to choose the timeout $t_{\mathcal{H}}$ to be t'. However, t' is a biased estimator of $t_{\mathcal{H}}$, and on many distributions in our sample data, t' *underestimates* $t_{\mathcal{H}}$. As an example, consider an inter-arrival time distribution $\mathcal{D}$ with the distribution function $f_{\mathcal{D}}$ pictured in Figure 2.

The best timeout $t_{\mathcal{D}}$ for $\mathcal{D}$ is at the end of the peak, at time 1/2. However, the first few inter-arrival times drawn from $\mathcal{D}$ are likely to be around 1/4, so at first t' will be around 1/4. This means that using t' as the timeout would have the costly result of the circuit initially being reopened more often than necessary.

There are standard methods to approximate the bias of a biased estimator, for example using bootstrapping [11]. However, in our case a simpler method of countering the bias of t' works well: to simply set the timeout to be t' plus a small correction,

$$t_{\mathcal{H}} = t' + \delta.$$

Experimentally we find that our results are quite insensitive to the value of δ, see Figure 6. Using $\delta = 0.1\times\mathcal{O}$ gives roughly a 5% improvement over $\delta = 0$, when $\mathcal{O}$ is large.

A last optimization is that, if $t' = 0$, then few small inter-arrival times have been observed, so the underestimation problem doesn't arise. In this case we do not want to increase $t_{\mathcal{H}}$, as the holding cost would increase unnecessarily. Hence if $t'=0$ we set $t_{\mathcal{H}} = 0$. Thus the formula for $t_{\mathcal{H}}$ is

$$\text{If } t'=0 \text{ then } t_{\mathcal{H}} = 0 \text{ else } t_{\mathcal{H}} = t' + 0.1\times\mathcal{O}.$$

Computing $t_{\mathcal{H}}$ is very efficient, and can be done in linear time using a single pass through the histogram.

On the arrival of the first packet in a conversation, no data are available about the distribution, but we must still set a timeout. Experimentally we find that the best choice of initial timeout is $0.1\times\mathcal{O}$, though again the results are insensitive to the exact value, see Figure 6. We have achieved some very small improvements over the empirical results described below by doing more detailed tuning. It is reasonable to expect that if necessary, further improvements could be achieved by careful tuning on a collection of representative data.

B. Sensitivity Analysis

The Lru-based and Adaptive policies have some parameters that need to be set. The Lru-based policy has the parameter c, while the Adaptive policy has parameters δ, the size of the histogram, the maximum timeout M, and the initial timeout after the first packet in a conversation. To determine the best values of these parameters, we varied each one individually, running the policies on the data sets from AT&T, UCB, USC and IIT Delhi. The average relative costs normalized by the optimal cost from these locations are plotted in Figures 5 through 7.

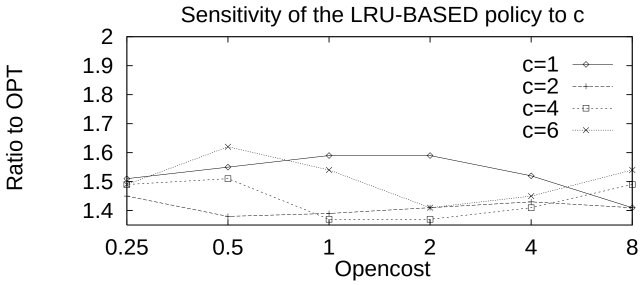

Fig. 5. Sensitivity of LRU-BASED policy to c in the holding cost model

We find that for the LRU-BASED policy, $c = 2$ works best, i.e. drop a conversation if it has been idle for $\mathcal{O}/2$ time units. We notice that $c = 4$ also performs well, so a value in the range 2 to 4 will be acceptable. This insensitivity to the precise value of c has a very nice consequence, namely that if $\mathcal{O}$ is known only imprecisely, the LRU-BASED policy will work well.

For the ADAPTIVE policy, the cost is quite insensitive to the precise values used for the parameters. There is remarkably little variation with changing histogram size, and a small histogram with 10 entries provides the full benefit of the policy with a negligible computational overhead. The best values for the other parameters are $\delta = 0.1$, $M = 1$, the initial timeout is $0.1 \times \mathcal{O}$, and these values are used in the evaluation below.

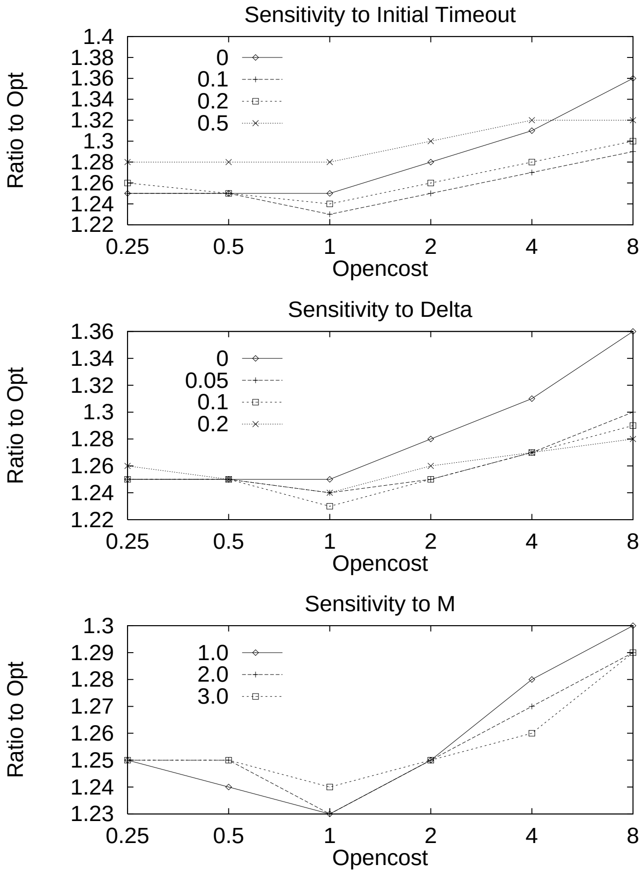

Fig. 6. Sensitivity of Normalized Performance to Policy Parameters in the Holding Cost Model.

C. Evaluation of Policies

We simulated the LRU-BASED policy, the MEAN-VARIANCE policy, and the ADAPTIVE policy on each of the traces. Each policy was run for values of the open-cost $\mathcal{O}$ ranging from 0.25 to 8. To be able to evaluate the policies across different values of $\mathcal{O}$ and different data sets, we normalize the cost by dividing by the cost of the optimal offline strategy (OPT) on the same data. The resulting normalized costs are plotted in Figure 8 for each of the original data sets. Each plot contains three lines, corresponding to the LRU-BASED policy, the MEAN-VARIANCE policy, and the ADAPTIVE policy using a histogram of size 10. Each line corresponds to an average over all traces from a site.

We observe that the MEAN-VARIANCE policy is almost always worse than the other two policies, while the ADAPTIVE policy is consistently better than the LRU-BASED policy. The only data point where ADAPTIVE is not doing better than LRU-BASED and MEAN-VARIANCE is at USC for $\mathcal{O} = 8$. This can be explained by noting that because of high traffic density, the USC traces are very short. Thus many conversations have only a few (2 or 3) packets, and the ADAPTIVE policy does not have enough time to learn the inter-arrival time distributions for such conversations.

To make a more comprehensive comparison of the policies, we took averages of the averages from each location. The result is shown in Table I. Each row corresponds to

TABLE I

NORMALIZED COSTS OF POLICIES IN THE HOLDING COST MODEL. EACH NUMBER IS THE AVERAGE OF THE AVERAGE COST IN EACH DATA SET.

	ADAPTIVE	LRU	MEAN-VARIANCE
8.00	1.30	1.41	1.53
4.00	1.28	1.43	1.57
2.00	1.25	1.42	1.60
1.00	1.23	1.39	1.58
0.50	1.24	1.39	1.55
0.25	1.25	1.45	1.49

a value of $\mathcal{O}$. We note first that the LRU-BASED policy is consistently better than the MEAN-VARIANCE policy. This result was somewhat surprising, and we tried some variations in the MEAN-VARIANCE policy's use of the mean and deviation, but without much improvement. Therefore it seems that the mean and deviation alone do not give enough information about the inter-arrival time distributions to design a good holding policy in this model.

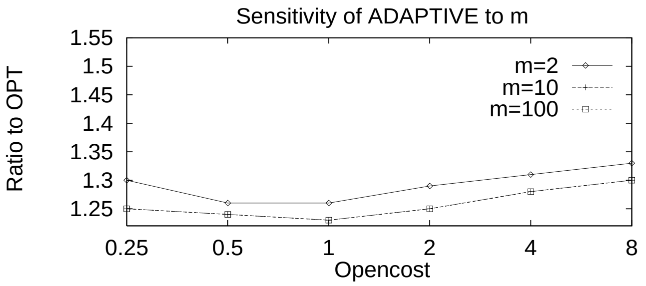

Fig. 7. Sensitivity of Normalized Performance to Histogram Size in the Holding Cost Model.

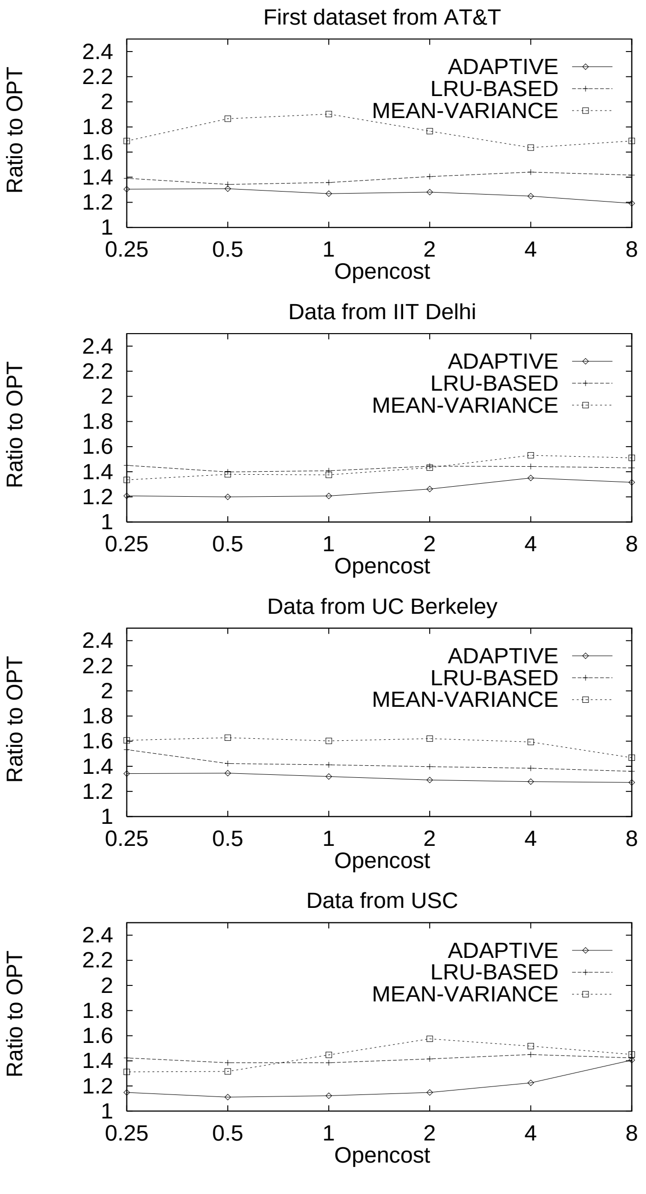

Fig. 8. Normalized performance of policies in the holding cost model for each of the original data sets.

Secondly we see that even the Adaptive policy is a significant improvement over the Lru-based policy, and is typically 35% closer to the optimal (non-causal) optimal than the Lru-based policy.

After finding the best values for the parameter of Lru and Adaptive on some of the data sets, we ran the policies on the rest of the data, namely the new data sets from AT&T and Yale. This is necessary to ensure that the parameters are good for all Internet data, not just on the data for which they were tuned. In Figure 9 we show the performance of the policies on the new data. The Mean-Variance policy performed significantly worse on these data, and these results are not shown. The performance of the Lru and Adaptive policies on this data is very similar to before, so we expect our conclusions to hold in general.

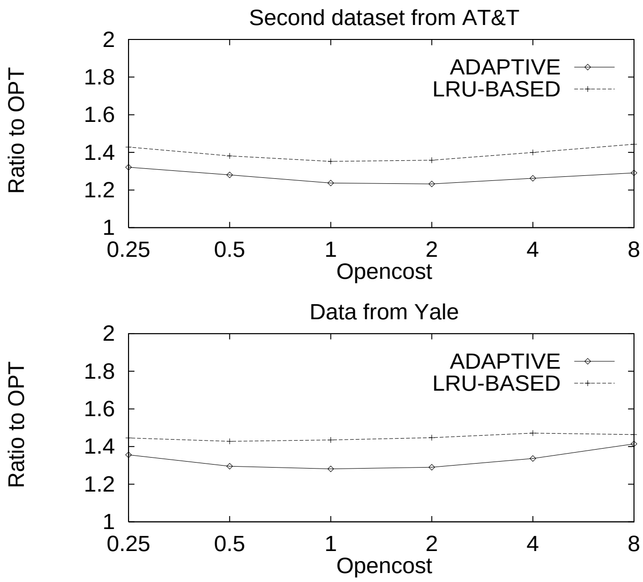

Fig. 9. Normalized performance of policies in the holding cost model for the new data sets.

VII. The Paging Pricing Model

In the second pricing model we study, once the user has paid for a block of connections to a site, there is a fixed charge for each call setup (including both network cost and loss of user utility during the call setup time). Our results are valid whether or not there is also a per-packet charge. We assume that there is no holding charge, so it makes sense to close a VC only when the fixed upper limit has been reached. For example, if a site is authorized to have at most 32 circuits open, and is charged for 32 circuits (as opposed to a higher charge for a site that is allowed 128 circuits), then the site would like to have all 32 circuits open at all times. This type of pricing model is common in X.25 networks.

A. Holding policies

When a packet arrives on a virtual circuit that is closed, the virtual circuit must be opened to transmit the packet.

Since in this pricing model there is a bound on the number of connections, an open virtual circuit may need to be closed. The holding policy must decide which of the currently open circuits is to be closed.

We consider several different policies in detail. As in the previous pricing model, there is great variation in the amount of information about the observed inter-arrival times used by the various policies.

The optimal (non-causal) strategy is to drop the conversation which will be inactive for the longest period of time. This is exactly the same as in the optimal page replacement algorithm in virtual memory systems.

A.1 The Random policy

One simple policy that uses absolutely no knowledge about the conversations themselves is to close one of the open conversations at random when a new one needs to be opened. We call this policy Random.

A.2 The Lru Policy

Since our traces indicated the presence of temporal locality of reference, it is natural to consider the Lru policy. This policy maintains an LRU reference stack, and closes the VC at the bottom of the stack when a new VC is needed. This policy uses no information about the inter-arrival time distributions, but does use information about recent packet arrivals.

A.3 The Mean-Variance policy

The Mean-Variance policy, like the Mean-Variance policy for the holding cost pricing model, uses an exponentially averaged mean, and an exponentially averaged mean deviation from this mean. Suppose the estimated average is μ and the current estimated deviation is σ. After each packet arrival on a VC, a timer is set which is equal to $\mu + 2\sigma$. If no reference has occurred to this VC in this period, the VC is marked eligible to be dropped. On a fault, we drop the eligible VC with the most elapsed time since the last packet arrival. In the case that no eligible VCs are present, we drop the VC with the largest remaining timer value.

A.4 The Adaptive Policy

Our Adaptive policy is based on the Median Algorithm of [4]. For each open circuit, our policy estimates the waiting time until the next packet and drops the open circuit with maximum estimate. The estimated waiting time for a circuit is just the median of the tail of its inter-arrival time distribution, defined precisely below. In [4], the Median Algorithm was proved effective in the following theoretical sense: if the inter-arrival time distributions remain unchanging over time, and the conversations are independent of each other, then the expected cost of the Median Algorithm is at most a factor of 5 greater than the best adaptive algorithm, regardless of the inter-arrival time distributions. Here we show that the Median Algorithm works even better in practice.

When $\mathcal{D}$ is known

Let $\mathcal{D}$ be the presumed underlying distribution on inter-arrival times for some circuit, and suppose that $\mathcal{D}$ has probability density function $f_{\mathcal{D}}$. For each circuit, we compute the estimated waiting time for the next packet given the amount of time, t, we have waited since the last packet. This estimated time, $T(t, \mathcal{D})$, is the median of the distribution after t, *i.e.* the least value of τ such that

$$\frac{\int_t^{\tau} f_{\mathcal{D}}\, dx}{\int_t^{\infty} f_{\mathcal{D}}\, dx} \geq 1/2.$$

We close the open circuit with largest $T(t, \mathcal{D})$. Note that we use only the tail of the distribution, since the initial part corresponds to time that has already passed since the last packet arrived.

When $\mathcal{D}$ must be learned

As in Section VI we need to gather empirical information on each distribution $\mathcal{D}$. For this pricing model, there is no natural choice for the last interval of inter-arrival times. Thus, instead of keeping a static histogram we use a dynamic histogram, in which the intervals change dynamically. A dynamic histogram $\mathcal{H}$ consists of a collection of m disjoint intervals, where each interval I consists of a minimum inter-arrival time, $\min_I$, a maximum inter-arrival time, $\max_I$, and a count, c_I, of the number of observed inter-arrival times in the interval $[\min_I, \max_I]$. We will describe how the intervals are maintained below.

Let Φ be the set of intervals with $\min_I \geq t$. We define the estimated time $ET(t, \mathcal{H})$ to the next arrival as the minimum τ such that

$$\frac{\sum_{I \in \Phi, \min_I \leq \tau} (\min_I - t) c_I}{\sum_{I \in \Phi} c_I} \geq 1/2.$$

When Φ is empty, and thus we have no data available about the distribution, we use the following rule. If only one packet has arrived on the conversation, we set $ET(t, \mathcal{H})$ to be $25t$, while if more than one packet has arrived on the conversation we set $ET(t, \mathcal{H})$ to be t. This tends to quickly close circuits where we have only observed a single packet. The value 25 for the initial median estimate multiplier was optimized for the sample data, but its exact value is not very important, see Figure 11 below.

Our policy closes the open circuit with largest value of $ET(t, \mathcal{H})$. The value of $ET(t, \mathcal{H})$ can be easily computed in time linear in m, using a single pass through the histogram $\mathcal{H}$.

Maintaining a Dynamic Histogram

Each time a new packet arrives on a circuit, if the inter-arrival time belongs to some interval I in $\mathcal{H}$ then c_I is incremented. If no such interval exists, a new interval I is created with $\min_I$ and $\max_I$ equal to the inter-arrival time and $c_I = 1$. When the number of intervals exceeds m some intervals are merged. The choice of the parameter m is not critical. Experiments suggest that a good choice for m is between 10 and 100 (see Section VII-C).

The merging of intervals is done as follows. Let $n(\mathcal{H})$ be the number of inter-arrival times observed so far. For each interval I such that $c_I < \frac{2n(\mathcal{H})}{m}$ we merge I with its closest neighboring interval. We continue merging until no such interval exists.

B. Sensitivity Analysis

The Adaptive policy has two parameters that need to be set: the maximum histogram size, and the initial median estimate multiplier. To determine the best values of these parameters, we varied each one individually, running the policies on the data sets from AT&T, UCB, USC and IIT Delhi. The average relative costs from these locations are plotted in Figures 10 and 11.

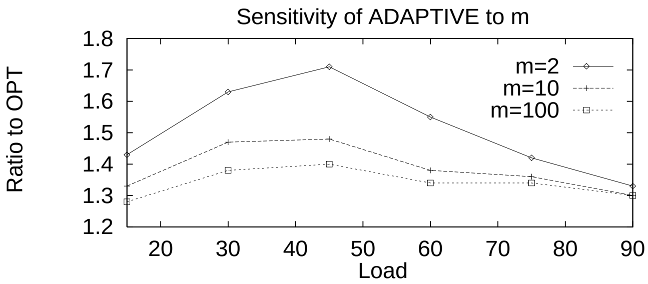

Fig. 10. Sensitivity of the Adaptive policy to maximum histogram size in the Paging Model.

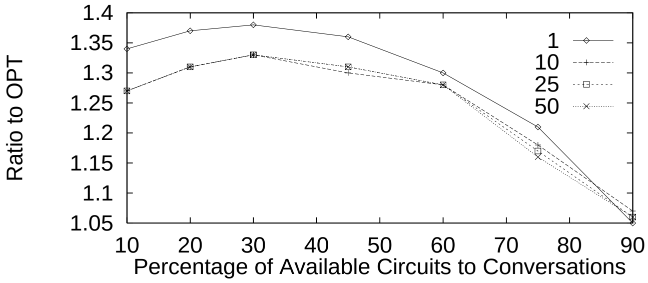

Fig. 11. Sensitivity of the Adaptive policy to Initial Median Estimate Multiplier in the Paging Model.

Notice that the performance of the Adaptive algorithm is relatively insensitive to choice of the maximum histogram size, with the maximum size of 10 giving nearly as good performance as 100. Notice also, that the performance of Adaptive is relatively insensitive to the precise value of the initial median estimate multiplier, with values between 10 and 50 giving nearly identical performance. In the evaluation below we have set the maximum histogram size to 100 and the initial median estimate multiplier to 25.

C. Evaluation of Policies

We simulated the Random policy, the Mean-Variance policy, the Lru policy, and the Adaptive policy on the datasets. Each trace has packets from a number of connections. However, some conversations may be active at different times in the trace, so we define the *active conversation number* of a trace to be the maximum over t of the number of conversations that have packets both before and after t in the trace. The policies were simulated with the ratio of the maximum number of open circuits to the active conversation number of the trace ranging from 15% to 90%. We normalize the costs by dividing by the cost of the optimal offline strategy (Opt) on the same data. The resulting normalized costs are plotted in Figure 12. Each line corresponds to an average over all traces from a site.

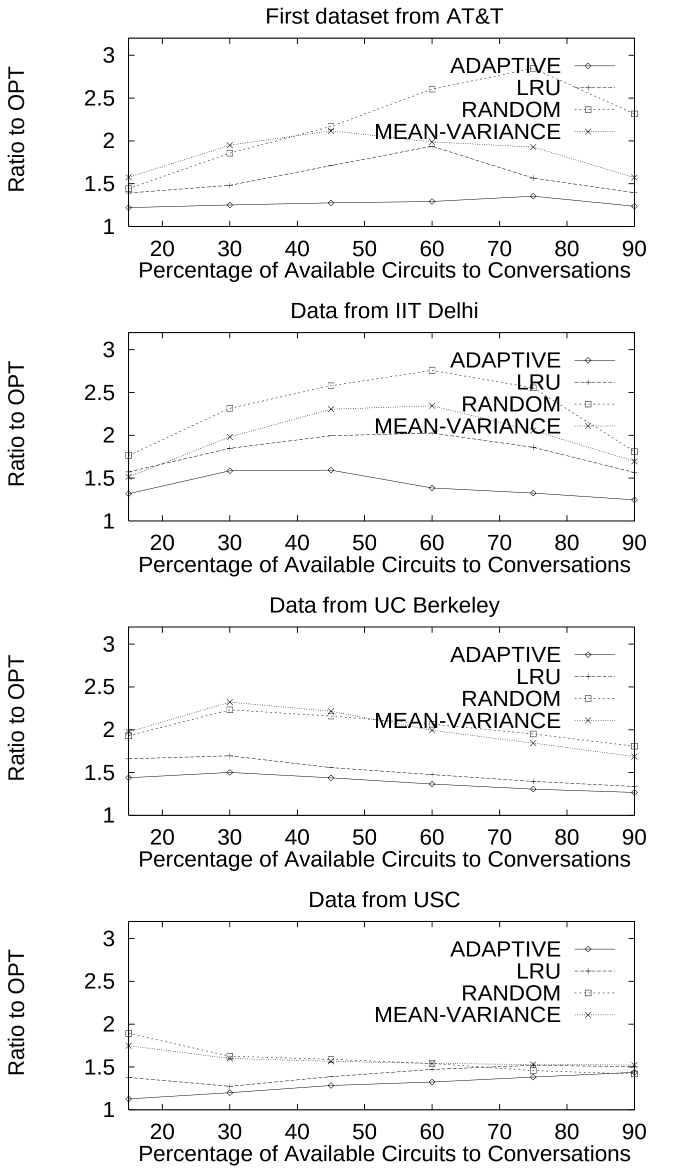

Fig. 12. Normalized performance of the strategies in the paging model for each of the original data sets.

As can be seen from Figure 12, Lru is consistently better than Mean-Variance and Random. Surprisingly, the Mean-Variance and Random strategies are comparable in their performance. The ADAPTIVE policy is consistently better than LRU.

To make a more comprehensive comparison of the policies, we took the average of all the location averages. The result is shown in Table II. Each row corresponds to the number of available virtual circuits as a percentage of the active conversation number.

TABLE II

NORMALIZED COSTS OF POLICIES IN THE PAGING MODEL. EACH NUMBER IS THE AVERAGE OF THE AVERAGE COST IN EACH DATA SET.

	ADAPTIVE	LRU	MEAN-VARIANCE	RANDOM
15%	1.28	1.50	1.70	1.76
30%	1.38	1.57	1.96	2.01
45%	1.40	1.66	2.05	2.12
60%	1.34	1.73	1.97	2.24
75%	1.34	1.59	1.84	2.20
90%	1.30	1.45	1.62	1.84

After setting the initial median estimate multiplier to 25 and the maximum histogram size to 100, we ran the ADAPTIVE policy on the new data sets from AT&T and Yale. In Figure 13 we show the performance of the ADAPTIVE policy on the new data. The ADAPTIVE policy has similar performance on this data, so we expect the policy to work in general and not just for the originally collected data.

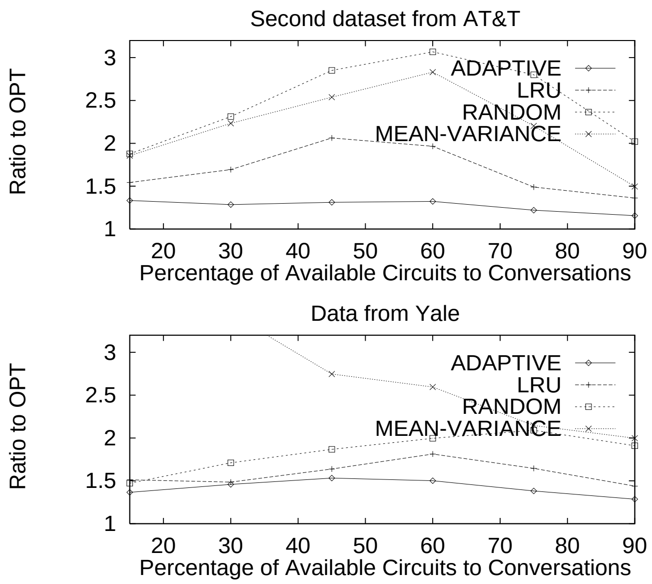

Fig. 13. Normalized performance of the strategies in the paging model for each of the new data sets.

The extremely poor performance of MEAN-VARIANCE is quite surprising. We investigated to check whether the prediction strategy was doing a reasonable job. We found that $62-69\%$ of the inter-arrival gaps lie in the interval $[\mu-\sigma, \mu+\sigma]$ and $78-87\%$ of the inter-arrival gaps lie in the interval $[\mu-2\sigma, \mu+2\sigma]$. Thus, the packet-arrival pattern does seem to fit the model assumed by MEAN-VARIANCE. The problem is that while in a cluster of closely arriving packets, MEAN-VARIANCE predicts successive arrivals well, but when a large gap occurs, MEAN-VARIANCE does poorly, since it has tuned its parameters to the preceding burst. As such, even a small gap after a very high rate burst causes a timeout whereas a larger gap after a medium-rate burst does not cause a timeout.

VIII. SYSTEM CONSIDERATIONS

In this section, we describe some system considerations in implementing holding time policies. We expect a holding time policy to be implemented in dual-ported routers that link IP and ATM networks. In such routers, the arrival of an IP packet triggers a search of a VCI cache, and if no VCI is found, a signaling entity is invoked to establish a new virtual circuit. The new VC, when established, is placed in the VCI cache. Subsequent datagram arrivals result in VCI cache hits, and the datagram is forwarded to the ATM device driver along with the appropriate VCI. We now discuss how this picture is modified by the LRU and ADAPTIVE holding time policies for each of the two pricing schemes. We evaluate the additional instruction and memory cost of the two policies.

A. Holding cost pricing model

For both LRU and ADAPTIVE, after mapping a datagram to a VCI, the current timer for that VCI must be cleared and a new timer set. This timer is based either on a system wide timeout (LRU-BASED), or on per-VC information (ADAPTIVE). For ADAPTIVE, the inter-arrival histogram also needs to be updated. On a timeout, the signaling entity has to be notified, and the corresponding virtual circuit torn down.

An efficient way to implement a timeout is using a calendar queue [12], a data structure that consists of an array of *days* where each day is a doubly linked list of events. On a clock tick, a pointer advances to the next day and the associated actions are taken. To set a timer, an event is added to the corresponding day queue. By rounding off timeout values to one day, the cost of setting or clearing a timer is a small constant number of instructions, since the operations are simply to unlink and link elements from the list. Thus, we believe that setting and clearing timeouts has little overhead.

The ADAPTIVE policy has the additional overhead of updating a histogram. This involves a comparison and addition step, followed by a scan through the array to calculate the new timeout value. For a histogram of size 10, which we have found to be adequate, this cost is around 50 instructions. However, note that these actions do not need to be in the packet forwarding path: they can be initiated after handing the datagram over to the ATM device driver. This minimizes the effect of the holding time policy on the packet forwarding delay.

The LRU-BASED and ADAPTIVE schemes require differing amounts of state space. Let there be at most N active virtual circuits. Then, for both schemes, the calendar queue should be large enough to accommodate N events, where an *event* consists of a function pointer, a *next* and a *previous* pointer, typically 32 bits each. For the LRU-BASED scheme, we need an additional $\log N$ bits per VC to point to the the corresponding timer event, so that clearing the timeout can be done in constant time. For the adaptive scheme, we need an additional $m \log P$ bits per VC, where m is the number of buckets in the histogram, and P is the largest number of packets in a bucket. In order to adapt to changing conditions on a larger time scale, we should allow the histogram to change with time. The simplest method is simply to occasionally divide each histogram entry by two, say when the histogram contains 256 packets. An alternative would be to keep two histograms H and H_{new} for each VCI, as described for the paging model below.

To summarize, the state overhead for the LRU-BASED policy is $N(96 + logN)$ bits, and for ADAPTIVE is $N(96 + logN + mlogP)$ bits. Using typical values of N as 2K VCIs, m as 10 and P as 256, the corresponding state requirement for the LRU-BASED scheme (including the calendar queue overhead) is 26.8 KBytes, and for the adaptive scheme is 46.8 KBytes. Given the consistent gains from the ADAPTIVE scheme, we feel that the extra memory overhead is insignificant.

B. Paging model

For the LRU scheme, on a packet arrival, the corresponding VCI has to be pulled to the top of the LRU stack. When a page fault occurs, the bottom element of the LRU stack has to be dequeued. These are both constant time operations that take less than 10 instructions on typical RISC machines. If the stack is implemented as a doubly linked list, it requires $k(64 + \log k)$ bits of state, where k is the maximum allowed number of open circuits. For $k = 128$, this amounts to 1.1 KBytes.

For the ADAPTIVE scheme, on a packet arrival, we simply update the histogram. On a page fault, we need to compute the next packet's median expected arrival time for each VCI to determine the largest such value. The computation involves scanning the per-VCI histogram, as before, and when $m = 20$ we expect it to take around 100 instructions per VCI for every open VCI. This takes about $100k$ instructions (about 0.5 ms on a 30 MIPS machine, for $k = 128$). This is too long a time, since in the worst case, page faults could occur more closely spaced than 0.5 ms. This is unacceptable even if the computation is not on the packet forwarding path. This problem can be solved using the following variation. When ADAPTIVE has computed the median for each VCI, it can keep a *hit list* of VCIs in decreasing order of median. Each time there is a page fault, the VCI at the head of the list is closed. When a packet arrives, its VCI should be removed from the list. Thus the computed medians can be used to make a number of decisions about which VCIs to close, during the time that the next set of medians is being computed. Using this method, the number of instructions which occurs on the packet forwarding path for each page fault is similar to the LRU scheme.

Note that the packet forwarding path on a page fault can be shortened, both for the LRU-BASED scheme and for ADAPTIVE, by keeping a single VCI unused, so that when a page fault occurs, the unused VCI can immediately be assigned to that conversation. After the packet is forwarded, a VC can then be torn down in order to set aside the next unused VCI.

Since ADAPTIVE keeps information about circuits for some time after they have been closed, there is an additional parameter N of the number of semi-active circuits, where the semi-active circuits are the N most recently used circuits. There is garbage collection on these circuits such that the number of semi-active circuits is bounded by N, using an LRU scheme. As in the holding cost model, the histograms should adapt to changing conditions on a larger time scale. We prefer to keep two histograms H and H_{new} for each VCI. The histogram H is the one used to compute the median. When a packet arrives, the inter-arrival time is put into both histograms. When H_{new} contains say 256 packets, we swap the names H_{new} and H and empty the new H_{new}. This way the median will be computed using data that is not too old. By using this scheme, P is at most 512, and the holding policy is made adaptive over longer time scales.

The state information per VCI is a thus a pair of histograms, where each bin consists of the max and min element in the bin and a counter of the number of elements in the bin. ADAPTIVE also needs to store the ordering for the *hit list* of open circuits. Thus we find that the ADAPTIVE scheme uses $2mN(\log P + 2T) + 32k + N(64 + \log N)$ bits of state, where T is the precision of the inter-arrival times. For $N = 2K$, $k = 128$, $m = 20$, $P = 512$ and $T = 10$, this amounts to 413.3 KBytes.

IX. Conclusions

We have studied the problem of how long to keep open a VCI opened to carry an IP datagram over an ATM network. We proposed a formal model for this problem and investigated two pricing schemes. For both pricing schemes we described the non-causal optimal holding policy and studied a number of non-adaptive and adaptive policies. In order to evaluate the policies we collected empirical data sets from Ethernet LANs located at sites around the world. We trained the policies on 4 data sets and evaluated them on 2 others.

We conclude that LRU/LRU-BASED policies do well in both pricing models. On the data we collected these policies have only 41.5% higher cost than the non-causal optimum in the holding cost model and 58.3% higher cost in the paging model. Further we found that the system costs for implementation are small; on the scale of 10 instruction per packet and the memory overhead is 1-27 KBytes for typical cases. On the other hand, the MEAN-VARIANCE policies that use information about the mean and standard deviation of the inter-arrival time distribution do surprisingly poorly (55.3% worse than optimal in the holding cost model and 85.7% worse than optimal in the paging model) in all the many variations that we tried. On the other hand the ADAPTIVE policies, that gather more information about the inter-arrival time distributions, do the best of the policies that we considered. They have only 25.8% higher cost than non-causal optimal in the holding cost model and only 34% higher cost in the paging model. The system costs for the ADAPTIVE policies are reasonable; in the holding cost model there are roughly 10 instruction per packet on the packet forwarding path and 100 off this path. In the paging model a page fault requires a longer computation outside the packet forwarding path; in typical cases this amounts to 1/2 ms on a 30 MIPS machine. This means that the scheme may not be as useful in situations where the average interval between page faults is very small. However the overhead may be alleviated by sharing the computation among a number of page faults. In both models the memory requirements are reasonable but somewhat larger than the LRU/LRU-BASED policies. In typical cases the ADAPTIVE policies uses 47-413 KBytes of memory.

Based on performance and systems cost we propose that the ADAPTIVE policies be used, except when memory is very scarce or in the paging model if page faults occur very frequently, in which case the LRU/LRU-BASED policies are good alternatives.

Lastly, we note that our policies for both pricing models may be useful in a wider context. In general terms, the holding cost model involves a resource that is intermittently used, and must be "open" to be used. There is a cost for opening it, and a cost for each time unit it remains open. This scenario describes many specific problems, for example disk management in portable computers: the "open cost" is the loss of utility to the user while spinning up the disk, while the holding cost corresponds to depletion of battery power. Similarly the paging model can be phrased more generally: a large number of entities are competing for the use of a scarce resource. This general model is also interesting in a variety of contexts. It is an interesting direction for further study to determine whether our adaptive methods can provide a performance benefit for related applications.

ACKNOWLEDGMENTS

We wish to thank Bruce Mah at UC Berkeley, Peter Danzig at the University of Southern California, Per Abrahamsen at AT&T Bell Laboratories, and Morrow Long at Yale University for help in collecting the datasets. Thanks also to Ramon Caceres for many helpful discussions and for good advice.

REFERENCES

[1] R. Caceres, "The Pyramid IP to X.25 protocol interface: Merging DDN and PDN approaches," in *Proc. Uniform*, (Washington, DC), 1987.

[2] H. Saran and S. Keshav, "An empirical evaluation of virtual circuit holding times in IP-over-ATM networks," in *Proc. Infocom*, 1994. To appear.

[3] C. Lund, N. Reingold, and S. J. Phillips, "Adaptive holding policies for IP over ATM networks." AT&T Bell Laboratories, Technical Memorandum 11272-940215-08TM, 1994.

[4] C. Lund, N. Reingold, and S. J. Phillips, "IP over connection-oriented networks and distributional paging," in *Proc. Foundations of Computer Science*, (Santa Fe, NM), 1994. to appear.

[5] B. R. Harita and I. M. Leslie, "Dynamic bandwidth management of primary rate ISDN to support ATM access," in *Proc. ACM SigComm*, pp. 197–210, 1989.

[6] R. Caceres, P. B. Danzig, S. Jamin, and D. J. Mitzel, "Characteristics of application conversations in TCP/IP wide-area internetworks," in *Proc. ACM SigComm*, September 1991.

[7] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," in *Proc. ACM Sigcomm*, pp. pp 257–268, 1994.

[8] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of ethernet traffic," in *Proc. ACM SigComm*, September 1993.

[9] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SigComm*, August 1988.

[10] A. R. Karlin, K. Li, M. S. Manasse, and S. Owicki, "Empirical studies of competitive spinning for a shared-memory muliprocessor," in *Proc. Thirteenth ACM Symp. on Operating System Principles*, (Pacific Grove, CA), p. 41, Oct. 1991.

[11] B. Efron and R. J. Tibshirani, *An Introduction to the Bootstrap*. New York: Chapman & Hall, 1993.

[12] A. R. Brown, "Calendar Queues: A Fast $O(1)$ Priority Queue Implementation for the Simulation Event Set Problem," *Communications ACM*, vol. 31, pp. 1220–1227, October 1988.